N89-21760

1988

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

SPACE SHUTTLE MAIN ENGINE NUMERICAL MODELING CODE MODIFICATIONS AND ANALYSIS

Prepared by:	John P. Ziebarth
Academic Rank:	Assistant Professor
University and Department:	University of Alabama in Huntsville Computer Science Department
NASA/MSFC:	
Laboratory:	Structures and Dynamics
Division:	Aerophysics
Branch:	Computational Fluid Dynamics
MSFC Colleagues:	Luke Schutzenhofer and Paul McConnaughey
Date:	August 17, 1988
Contract Number:	NGT 01-002-099 The University of Alabama

SPACE SHUTTLE MAIN ENGINE NUMERICAL MODELING CODE MODIFICATIONS AND ANALYSIS

by

John P. Ziebarth
Assistant Professor of Computer Science
University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

The user of Computational Fluid Dynamics (CFD) codes must be concerned with the accuracy and efficiency of the codes if they are to be used for timely design and analysis of complicated three-dimensional fluid flow configurations. A brief discussion of how accuracy and efficiency affect the CFD solution process is given. A more detailed discussion of how efficiency can be enhanced by using a few Cray Research Inc. utilities to address vectorization is presented and these utilities are applied to a three-dimensional Navier-Stokes CFD code (INS3D).

ACKNOWLEDGMENTS

I am grateful for the opportunity that I have had to participate in the Summer Faculty Fellowship Program. The work that I completed this summer would not have been possible without the advice and assistance of Luke Schutzenhofer and Paul McConnaughey.

I also appreciate the support of Dr. Carl Davis and the University of Alabama in Huntsville for allowing me to work with this program.

I would like to thank Dr. Mike Freeman for his encouragement and guidance throughout the program. The experience at Marshall Space Flight Center has been very enjoyable, and I am indebted to the National Aeronautics and Space Administration and the American Society for Engineering Education for providing me with the chance to work at NASA/Marshall Space Flight Center in Huntsville, Alabama.

LIST OF FIGURES

Figure 1	Computer Requirements for Computational Aerodynamics	XXXIV - 6
Figure 2	Loopmark Results for Original Version of Subroutine VISRHS2	XXXIV - 11
Figure 3	Loopmark Results for Modified Version of Subroutine VISRHS2	XXXIV - 12
Figure 4	Flowtrace Summary for Original Version of INS3D	XXXIV - 13
Figure 5	Flowtrace Summary for Modified Version of INS3D	XXXIV - 14
Figure 6	Flowtrace Calling Tree for INS3D	XXXIV - 15

INTRODUCTION

Computational Fluid Dynamics (CFD) has become an extensively used tool in the design and analysis of complex three-dimensional flows including those through the Space Shuttle Main Engine (SSME). Rapid advances in CFD over the last decade have provided to the user community a large set of computer codes, each with various capabilities and constraints. These codes are adopted for use by scientists and engineers who may or may not understand fully the physics and mathematics in the code. Once adopted, these codes are adapted to solve a variety of problems, which hopefully, are similar enough to the ones for which the code was written to be applicable to the physics in the code.

In addition to this accuracy consideration, is the concept of efficiency. For full three-dimensional calculations, the amount of supercomputer time necessary to reach a solution can be quite large. To code developers this large computational time may not be of much concern; however, for code users interested in design and/or analysis, solutions must be achievable in a reasonable amount of time. Unfortunately, most current university curricula for engineers and scientists do not contain any preparation on how to efficiently write FORTRAN or use current supercomputers.

How to measure and/or evaluate accuracy and efficiency is of current interest to the CFD community in general, and to the CFD Branch at NASA/MSFC in particular. In fact, the entire process of doing a CFD calculation is somehow affected by either accuracy or efficiency. The solution process can be divided into three major components: pre-processing, processing and post-processing. Pre-processing involves primarily the geometry modeling and the grid generation. Processing involves the equation modeling of the flow physics, the implementation of a numerical method to solve the equations, the FORTRAN coding of the numerical method and the running of the computer program. The post-processing takes the massive amount of data produced by the code and transforms it into a usable form by some visualization methods.

OBJECTIVES

The objectives of this work were to:

1) Consider accuracy and efficiency as it relates to CFD codes, and begin to evaluate and establish guidelines and criteria for CFD code users.

2) Consider efficiency with respect to reducing the amount of CPU time needed to reach a solution. Do this by applying some Cray Research Inc. utilities to a three-dimensional Navier-Stokes CFD code (INS3D).

ACCURACY

Accuracy is affected at all the levels of processing. In the pre-processing phase, a precise modeling of the geometry is necessary. Regardless of how this geometry modeling is done it quite often only approximates some complicated boundaries and corners of three-dimensional configurations. The grid generation is then done on the resulting geometry model by some mathematical method. It is well documented [1, 2] that the final solution is affected by the distribution of grid points in this computational domain; however, quantitative measures of how "good" a grid is are not readily available.

In the processing phase of the solution accuracy is affected by many factors. Although the Navier-Stokes equations are generally accepted as a full description of turbulent fluid motion in a continuum, the complexity of the equations and the extremely small time and length scales of turbulent motion prohibit practical numerical computation of turbulent flows by this method. Thus, many levels of approximation are used. These include both linear and nonlinear inviscid, boundary layer, Reynolds averaged Navier-Stokes and large eddy simulation approximations. Once a set of equations is chosen to solve the flow physics, a numerical method to solve these equations must be implemented. Thus, accuracy has been affected at two levels in the processing so far. The next step is the coding (typically in FORTRAN) and the running of the code. Here accuracy is dependent on correct coding, precision of the computer system, and degree to which the solution is allowed to converge.

In the post-processing, accuracy can be affected by the visualization method used. Visualization of the flow involves taking the computed data and inputting it into a software package which can be either a commercial or locally developed product and outputting the results onto some type of graphics hardware. Thus, correctly written software is required and the resolution of the output device medium makes a difference in the accuracy of the visualization.

Currently those involved in CFD approach accuracy from different directions. Some accept the "answers" as produced by existing codes as being reasonable while others tend to be skeptical of at least some aspects of the solution process. Probably the most accepted components

are the geometry modeling and the post-processing. The grid is often thought to be acceptable if a flow solution can be arrived at by using it. Most doubt, if any exists, is usually directed at the flow solution itself. Unfortunately, questions concerning the accuracy of the flow solution do not necessarily have simple solutions. Many factors play a part in the solution; the modeling of the equations, the numerical method used to solve these equations, the convergence criteria used, boundary condition implementation, turbulence modeling, grid dependencies, correct coding, etc.

EFFICIENCY

The efficiency of a CFD solution also affects pre-processing, processing, and post-processing. This is especially true in design/analysis environments and in situations where computational resources are scarce. Pre-processing has typically been the most time consuming part of the solution process, especially when an analysis is being done on a new configuration. Current efforts in interactive geometry modeling and grid generation have helped some, but this phase is still an area of ongoing research and development. In the processing phase efficiency is tightly coupled to computational resources. An interesting projection [3] was made which indicates that to be useful in design, computational aerodynamics requires machines capable of at least one trillion floating-point operations per second (see Fig. 1). Current supercomputers do not yet meet this requirement. Post-processing has not been a particularly inefficient part of the entire solution process if all that is required is inputting the solution values to a software package and then displaying the results on a graphics workstation. However, this is a time consuming operation if real time high resolution graphical animation of the flow is desired. This can be a very computationally intensive task and can involve the transmission of extremely large quantities of data.

The concern of this current work is efficiency during the processing phase. We assume the pre-processing has been done in what follows and no post-processing is discussed. Current CFD users are constrained in achieving a solution by the following factors:

a) The CPU time needed to reach an acceptable level of convergence may be large (greater than one hour).
b) Supercomputing centers are often saturated, thus, the CPU time translates to a much larger wall clock time.
c) Most or all of the CFD code is typically written by a researcher who often is not concerned with or perhaps even knowledgeable of how to write efficient FORTRAN code.

A CFD user can improve efficiency in varicus ways. One way is to modify the code so that the FORTRAN is more efficient. This is typically referred to as optimization and is not related to vectorization. Optimization improves efficiency even on a scalar processor. Although writing

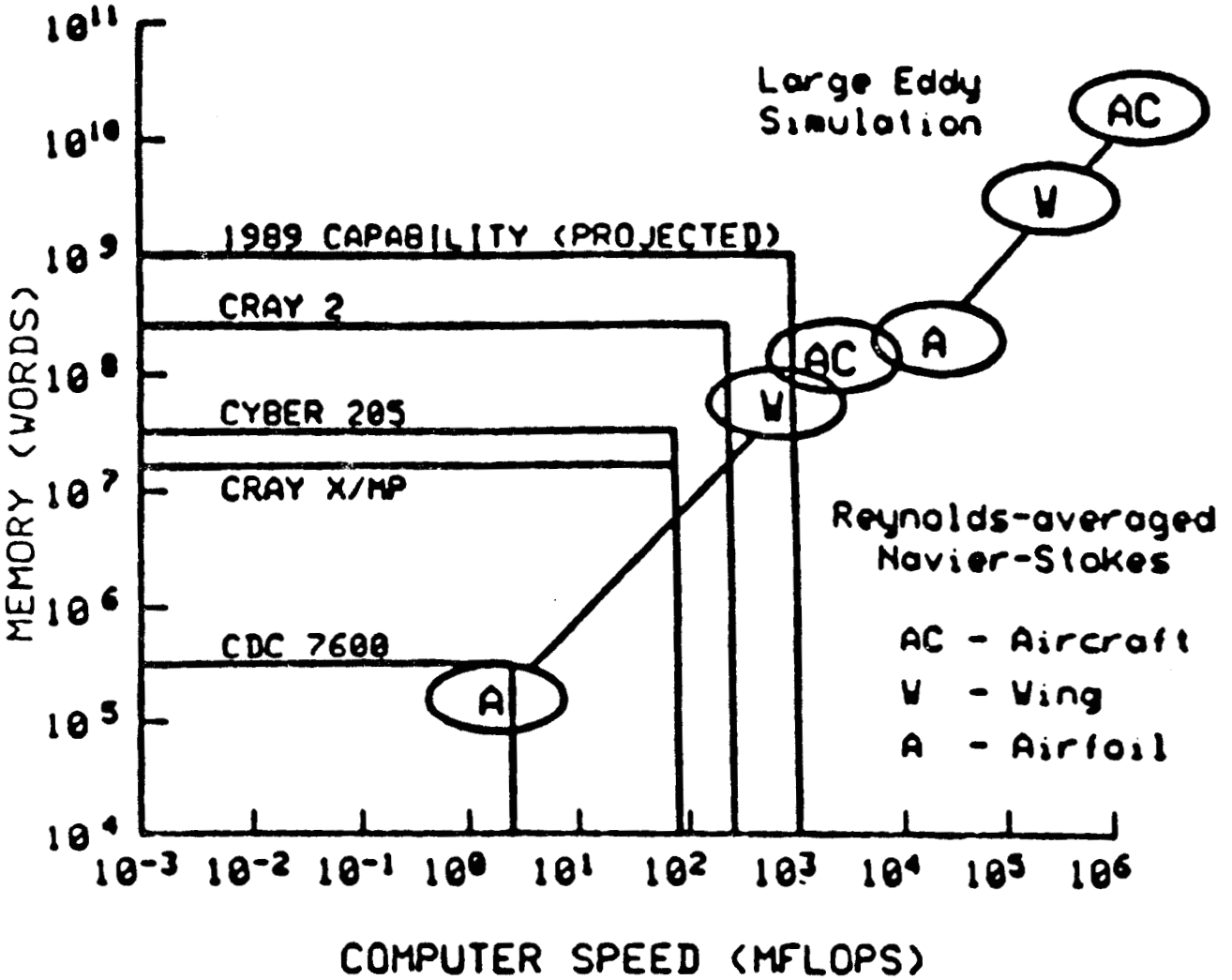

Figure 1 Computer Requirements for Computational Aerodynamics

efficient optimized code is best done during the initial code development, it is usually neglected at that time for two reasons; 1) getting a solution is of a higher priority than is writing efficient code and 2) many CFD researchers do not know how to write efficient FORTRAN code. For the user, optimizing the code can be very time consuming, requiring a major rewriting of much of the code. Methods for optimization are discussed in [4].

Vectorization is a capability of current supercomputers which yields the greatest efficiency benefit. It essentially is parallel processing since it implies that a single instruction performs many operations instead of just one; however, in the context of this report, parallel processing and vectorization will be discussed separately. More information on vectorization can be found in [4].

Fortunately for CFD users vectorization is a capability automatically implemented by the FORTRAN compilers of current supercomputers. These compilers have matured over the last few years to the point where they do a pretty good job, and they should continue to improve with time. Most CFD users currently depend on this automatic capability and actually never know which loops (FORTRAN DO loops) actually do or do not vectorize. Two capabilities of the Cray FORTRAN compiler will be discussed and their effect on a three-dimensional incompressible Navier-Stokes code will be described.

Before continuing with the vectorization discussion a few comments on parallel processing are appropriate. Parallel processing, also called multitasking (at least by Cray Research Inc.), is the capability to divide the solution into segments such that they are being done concurrently on more than one CPU. Multitasking is further divided into two parts, macrotasking and microtasking. Macrotasking refers to executing multiple segments of a program simultaneously through library calls. This requires restructuring of the code by the user and is generally not a trivial task. Microtasking refers to being able to simultaneously execute segments of a program at the DO loop level through compiler directives. This is generally easier to do than macrotasking, creates less overhead, and produces a code which from the computer system management point of view is very nice because it runs even if only one CPU is available. If and when another CPU becomes available the microtasked code has the ability to make use of the free CPU (or CPUs). Implementing microtasking is important for the CFD user although typically it is not done. This is primarily because it is a relatively new feature and little effort has been made to train CFD users to make use of it.

Fortunately for users, compilers in the future will be able to do at least some microtasking automatically. Cray should eventually release a feature called autotasking, which will automatically produce multitasked code for certain program structures.

An important point for users to be aware of is that multitasking reduces wall clock time, whereas vectorization and optimization reduce CPU clock time. The user should always reduce CPU time first by all methods available then reduce wall clock time through multitasking.

VECTORIZATION TOOLS AND INS3D

Two of the UNICOS utilities available from Cray Research Inc. for their Cray X-MP computer system are loopmark and flowtrace. These utilities are available under Cray's CFT77 compiler. A warning is in order for users of Cray computer systems. Cray offers for their X-MP systems both COS and UNICOS operating systems and various compilers (or versions of compilers). Consult the manuals for the system being used to see which utilities are available and how to implement them. Also, over the last decade compilers have continued to get "smarter", so FORTRAN code may run differently and vectorization may be applied differently now than in the past. Be aware of this when comparing results.

The loopmark and flowtrace utilities will be applied and results discussed for INS3D. The geometrical configuration considered is three-dimensional flow past a circular cylinder between two parallel plates. See [5] for a discussion of this case.

The loopmark utility can be used to determine which inner DO loops were automatically vectorized by the compiler. The listing will bracket the DO loops and will indicate which vectorized and which did not (see Figs. 2 and 3). Loopmark also furnishes a reason for the loop not vectorizing. Knowledge of what causes and what inhibits vectorization [4] is then necessary so that the code can be modified to try and implement vectorization in loops where the compiler could not. This involves a restructuring of the code. Care must be taken to be sure the code runs the same way and yields identical results before and after the restructuring. This typically means only a few changes should be implemented between check runs of the code.

Loopmark creates a listing file which may be much longer than the original code. Figures 2 and 3 are only small portions of the original and modified subroutine VISRHS2 and are only intended to show the structure of the listing. The INS3D code is approximately 5570 lines long and subroutine VISRHS2 is 650 lines long, so Figures 2 and 3 only show the format of the output. The left column in each figure shows sequential line numbers with two dots inserted for missing code. Figure 2 shows a portion of VISRHS2 where three nested DO loops begin. The DO 50 loop is a candidate for vectorization. Figure 3 shows VISRHS2 restructured so the original inner DO loop (DO 50 in Fig.

2) now is broken into four shorter DO loops (DO 50, DO 51, DO 52 and DO 53), and each of these loops does vectorize. The 50 CONTINUE in Figure 2 and the 53 CONTINUE in Figure 3 represent the same location in the code. Also included in Figures 2 and 3 is the vectorization information which is a part of the output of loopmark.

Flowtrace is a utility which will monitor calls to and from routines in the code and print various statistics about total execution time (see Figs. 4 and 5). In Figures 4 and 5 the columns from left to right are the routines called, the execution time in seconds for each, the percentage that each routine used of the total execution time, the number of times the routine was called, the average time (number of times called divided by execution time for routine) and finally the calling program unit for each routine. Flowtrace also prints out a calling tree (Fig. 6) for the program. The indentations of the routines in the tree indicate the levels of depth in the tree.

Flowtrace does add overhead to the program run but it is a very useful utility. It can also be enabled only for parts of a program thus creating less total overhead. When using flowtrace pay attention to the percent column. Man hours (or days) should probably not be spent rewriting and restructuring sections of the code where little time is spent. Also realize that as a certain section of the code is speeded up, its execution time and percentage of total time will decrease, but the percentage of another section of the code will increase since the percentages must always sum to 100.

Figure 4 indicates that subroutine VISRHS2 consumes most of the CPU time and this represents the largest percentage of total time relative to any other subroutine. By looking first at the flowtrace output and then at the compiled code with loopmark enabled (Fig. 2) it is noted that only a small portion of the DO loops in VISRHS2 vectorize automatically. By restructuring of the code all inner DO loops in VISRHS2 can be vectorized and now VISRHS2 consumes 24.3 percent (Fig. 5) of the total time rather than the 36.2 percent seen in Figure 4 and the execution time for the subroutine is reduced from 690 seconds to 389 seconds. Thus, by simply restructuring one subroutine a speedup of 1.2 times can be attained. The subroutine VISRHS2 which is now vectorized still requires the most time and thus is a good candidate for microtasking of the triply nested DO loops in it.

```
4068      3.                C*****************************************************************
4069      4.                                  SUBROUTINE VISRHS2
4070      5.                C*****************************************************************
  .
  .
4107     42.                C
4108     43. S-----------<        DO 100 K=KEND2,KENDM
4109     44. :   S-------<        DO 100 L=2,LM
4110     45. :   S        C
  .
  .
4114     49. :   S                IKL=(K-1)*KK+(L-1)*LL
4115     50. :   S        C
4116     51. :   :   S---<        DO 50 J=1,JMAX
4117     52. :   :   S            IJL=J+(L-1)*LL
4118     53. :   :   S            I=IKL+J
4119     54. :   :   S      C-----------------------------------------
4120     55. :   :   S      C...CALCULATE XK, YK, ZK
4121     56. :   :   S      C-----------------------------------------
  .
  .
4228    163. :   :   S      C        A(2,3,J) = VNUTJ* (XSIX**2 + XSIY**2 + XSIZ**2)
4229    164. :   :   S              A(3,3,J) = VNUTJ* (XSIX*ETAX + XSIY*ETAY + XSIZ*ETAZ)
4230    165. :   :   S              A(4,3,J) = VNUTJ* (XSIX*ZETAX + XSIY*ZETAY + XSIZ*ZETAZ)
4231    166. :   :   S--->    50 CONTINUE
4232    167. :   S        C=================================================================
  .
  .
                                  V E C T O R I Z A T I O N   I N F O R M A T I O N
                                  -------------------------------------------------

*** *** Loop starting at line 51 was not vectorized because
        a value is defined in a conditionally executed block but used in another block of the loop
```

Figure 2 Loopmark Results for Original Version of Subroutine VISRHS2

```
4068      3.                C*****************************************************************
4069      4.                                    SUBROUTINE VISRHS2
4070      5.                C*****************************************************************
  .
  .
4111     46. S------------<       DO 100 K=KEND2,KENDM
4112     47. :   S--------<       DO 100 L=2,LM
4113     48. :   S        C
  .
  .
4117     52. :   S                IKL=(K-1)*KK+(L-1)*LL
4118     53. :   S        C
4119     54. :   S                IF (KPERI .EQ. 1) then
4120     55. :   :   V----<       DO 50 J=1,JMAX
4121     56. :   :   V            IJL=J+(L-1)*LL
  .
  .
4130     65. :   :   V--->    50 continue
4131     66. :   S        c
4132     67. :   S                else
4133     68. :   S        c
4134     69. :   :   V----<       DO 51 J=1,JMAX
4135     70. :   :   V            IJL=J+(L-1)*LL
  .
  .
4147     82. :   :   V                    ZKnn(j) = ( Z(IRR)- 8.*(Z(IR)-Z(IP))- Z(IPP) )/12.
4148     83. :   :   V--->    51 continue
4149     84. :   S        c
4150     85. :   S                endif
4151     86. :   S        c
4152     87. :   :   V----<       DO 52 j=1,JMAX
4153     88. :   :   V            IJL=J+(L-1)*LL
  .
  .
4176    111. :   :   V            endif
4177    112. :   :   V--->    52 continue
4178    113. :   S        c
4179    114. :   :   V----<       do 53 j = 1,jmax
4180    115. :   :   V            IJL=J+(L-1)*LL
  .
  .
4218    153. :   :   V              A(4,3,J) = VNUTJ* (XSIX*ZETAX + XSIY*ZETAY + XSIZ*ZETAZ)
4219    154. :   :   V--->    53 continue
4220    155. :   S        C
4221    156. :   S        C-----------------------------------------------------------------------
  .
  .
                              V E C T O R I Z A T I O N   I N F O R M A T I O N
                              ------------------------------------------------

*** *** Loop starting at line 55 was vectorized
*** *** Loop starting at line 69 was vectorized
*** *** Loop starting at line 87 was vectorized
*** *** Loop starting at line 114 was vectorized
```

Figure 3 Loopmark Results for Modified Version of Subroutine VISRHS2

ORIGINAL PAGE IS
OF POOR QUALITY

```
1 F L O W T R A C E  --  Alphabetized summary
0     Routine            Time executing  Called  Average T
  26 BC                 2.953   (  0.15%)    600        0.005 @02534712a Called by      MAIN
  14 COMET            102.060   (  5.36%)4453200            > @02540467a Called by      STEP
  20 ETAINV            47.676   (  2.50%) 19800        0.002 @02561161a Called by      STEP
   9 FLUXVE            46.220   (  2.43%)2226600            > @02541716a Called by       RHS
   3 GRID               0.016   (  0.00%)      1        0.016 @02541775a Called by    INITIA
   5 IC                 0.006   (  0.00%)      1        0.006 @02542222a Called by    INITIA
   2 INITIA                 >   (  0.00%)      1            > @02542313a Called by      MAIN
   4 JACOB              0.257   (  0.01%)      1        0.257 @02542452a Called by    INITIA
   1 MAIN               0.025   (  0.00%)      1        0.025 @02534546a Called by
   8 METRIC           111.373   (  5.84%)4453200            > Called by      RHS      STEP
     @02543141a                                                             2226600  2226600
  28 OUTPUT             0.032   (  0.00%)     12        0.003 @02545674a Called by      MAIN
   7 RHS              129.922   (  6.82%)    600        0.217 @02552200a Called by      STEP
  12 SMOOTH            60.761   (  3.19%)    600        0.101 @02552772a Called by       RHS
   6 STEP             213.562   ( 11.21%)    600        0.356 @02553733a Called by      MAIN
  19 TK               122.022   (  6.40%)2226600            > @02554776a Called by      STEP
  15 TKINV            159.074   (  8.35%)2226600            > @02555201a Called by      STEP
  18 TRI               14.113   (  0.74%)  72000            > Called by    XIINV  ZETAINV
     @02555457a                                                              32400    39600
  17 TRI2              10.001   (  0.52%)  36000            > Called by    XIINV  ZETAINV
     @02555571a                                                              16200    19800
  22 TRIP              14.930   (  0.78%)  39600            > @02555720a Called by    ETAINV
  21 TRIP2             10.346   (  0.54%)  19800            > @02556150a Called by    ETAINV
  27 VISCT              0.007   (  0.00%)    600            > @02556434a Called by      MAIN
  10 VISRHS            86.818   (  4.56%)2226600            > @02556442a Called by       RHS
  11 VISRHS2          690.153   ( 36.21%)    600        1.150 @02556621a Called by       RHS
  16 XIINV             41.489   (  2.18%)  16200        0.003 @02560350a Called by      STEP
  23 ZETAINV           41.989   (  2.20%)  19800        0.002 @02562424a Called by      STEP
 * * * TOTAL         1905.804   18039617 Total calls
```

Figure 4 Flowtrace Summary for Original Version of INS3D

```
1 F L O W T R A C E  --  Alphabetized summary
0     Routine            Time executing  Called  Average T
  26 BC                2.950   (  0.18%)     600        0.005 @02542663a Called by     MAIN
  14 COMET           101.112   (  6.32%)4453200            > @02546440a Called by     STEP
  20 ETAINV           47.241   (  2.95%)  19800        0.002 @02570143a Called by     STEP
   9 FLUXVE           46.727   (  2.92%)2226600            > @02547667a Called by      RHS
   3 GRID              0.015   (  0.00%)       1        0.015 @02547746a Called by   INITIA
   5 IC                0.005   (  0.00%)       1        0.005 @02550173a Called by   INITIA
   2 INITIA                >   (  0.00%)       1            > @02550264a Called by     MAIN
   4 JACOB             0.257   (  0.02%)       1        0.257 @02550423a Called by   INITIA
   1 MAIN              0.025   (  0.00%)       1        0.025 @02542517a Called by
   8 METRIC          111.038   (  6.94%)4453200            > Called by      RHS     STEP
     @02551112a                                                           2226600  2226600
  28 OUTPUT            0.033   (  0.00%)      12        0.003 @02553845a Called by     MAIN
   7 RHS             129.301   (  8.09%)     600        0.216 @02560151a Called by     STEP
  12 SMOOTH           60.290   (  3.77%)     600        0.100 @02560743a Called by      RHS
   6 STEP            212.457   ( 13.29%)     600        0.354 @02561704a Called by     MAIN
  19 TK              121.543   (  7.60%)2226600            > @02562747a Called by     STEP
  15 TKINV           158.434   (  9.91%)2226600            > @02563152a Called by     STEP
  18 TRI              14.025   (  0.88%)  72000            > Called by    XIINV  ZETAINV
     @02563430a                                                           32400    39600
  17 TRI2              9.905   (  0.62%)  36000            > Called by    XIINV  ZETAINV
     @02563542a                                                           16200    19800
  22 TRIP             14.784   (  0.92%)  39600            > @02563671a Called by   ETAINV
  21 TRIP2            10.240   (  0.64%)  19800            > @02564121a Called by   ETAINV
  27 VISCT             0.007   (  0.00%)     600            > @02564405a Called by     MAIN
  10 VISRHS           86.592   (  5.42%)2226600            > @02564413a Called by      RHS
  11 VISRHS2         388.858   ( 24.32%)     600        0.648 @02564572a Called by      RHS
  16 XIINV            41.285   (  2.58%)  16200        0.003 @02567332a Called by     STEP
  23 ZETAINV          41.762   (  2.61%)  19800        0.002 @02571406a Called by     STEP
* * * TOTAL         1598.885   18039617 Total calls
```

Figure 5 Flowtrace Summary for Modified Version of INS3D

```
1 F L O W T R A C E  --  Calling tree
      1   MAIN      02542517a
      2     INITIA    02550264a
      3       GRID      02547746a
      4       JACOB     02550423a
      5       IC        02550173a
      6     STEP      02561704a
      7       RHS       02560151a
      8         METRIC    02551112a
      9         FLUXVE    02547667a
     10         VISRHS    02564413a
     11         VISRHS2   02564572a
     12         SMOOTH    02560743a
     13       METRIC    02551112a
     14       COMET     02546440a
     15       TKINV     02563152a
     16       XIINV     02567332a
     17         TRI2      02563542a
     18         TRI       02563430a
     19       TK        02562747a
     20       ETAINV    02570143a
     21         TRIP2     02564121a
     22         TRIP      02563671a
     23       ZETAINV   02571406a
     24         TRI2      02563542a
     25         TRI       02563430a
     26     BC        02542663a
     27     VISCT     02564405a
     28     OUTPUT    02553645a
   STOP              in MAIN
```

Figure 6 Flowtrace Calling Tree for INS3D

CONCLUSIONS AND RECOMMENDATIONS

During the past decade CFD codes have been accepted with justifiable skepticism by users interested in design and analysis. These users are not interested in getting just a "number" out of these codes, they are interested in getting a believable "number" in an affordable amount of time. Today, with many codes promising good results, users must have criteria they can rely on to evaluate codes. They also must be able to efficiently apply these codes to various problems on often scarce or expensive computing resources.

CFD users need to be able to evaluate codes in a reasonable amount of time and with a high degree of assurance that the chosen codes are efficient and yield accurate results. Evaluation criteria should be established for this purpose. Along with establishing criteria to evaluate codes is the need to educate code users of the utilities and methods to develop efficient and accurate codes.

The application of the loopmark and flowtrace utilities to a CFD code show that efficiency can be enhanced by using basic tools available to but not necessarily known by users. Along with the development of evaluation criteria should be the education of users on how to develop efficient well-written FORTRAN code.

Besides loopmark and flowtrace are other utilities which can enhance performance of codes. Also, multitasking (parallel processing) needs to be used by CFD code users to make the most efficient use of supercomputer resources. Work is under way to apply both microtasking and macrotasking to INS3D; hopefully this work will be of interest to the user community.

REFERENCES

1. Thompson, J.F., Warsi, Z.U.A., and Mastin, C.W. Numerical Grid Generation: Foundations and Applications, North Holland, 1985.

2. Wang, T.S., Soni, B.K., "Goodness-of-Grid: Quantitative Measures", AIAA/ASME/SIAM/APS First National Fluid Dynamics Congress, July 25-28, 1988, Cincinnati, Ohio.

3. Peterson, V.L., "The Impact of Supercomputers on the Aerospace Sciences, "AIAA Twenty-fourth Aerospace Sciences Meeting, Reno, Nevada, January, 1986.

4. Ziebarth, J.P., "Introduction to Supercomputing", course notes, University of Alabama in Huntsville, 1988.

5. Rogers, S.E., Kwak, D., and Chang, J.L.C., "INS3D - An Incompressible Navier-Stokes Code in Generalized Three-Dimensional Coordinates", NASA Technical Memorandum 100012, November, 1987.